#4

PATENT APPLICATION
Attorney Doc. No. 2705-87

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of: Frank G. Bordonaro, Kui Zhang and Satyanarayana Rao Raparla

Serial No. 09/434,845 Group Art Unit: 2664

Filing Date: November 4, 1999 Examiner: Prenell P. Jones

For: METHOD AND APPARATUS FOR MEASURING NETWORK DATA PACKET DELAY, JITTER AND LOSS

Mail Stop Non Fee
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION OF KUI ZHANG UNDER RULE 37 C.F.R. 1.131

I, Kui Zhang, declare the following:

1. I am a co-inventor of the subject matter described in the present pending patent application entitled: METHOD AND APPARATUS FOR MEASURING NETWORK DATA PACKET DELAY, JITTER AND LOSS, Serial No. 09/434,845, filed November 4, 1999.

2. I currently work for Cisco Systems, Inc. My work mailing address is 170 West Tasman Drive, San Jose, CA 95134-1706.

3. Prior to January 19, 1999 (the filing date of U.S. Patent No. 6,397,359B1 to Chandra et al cited by the Examiner in connection with the present application), the other named co-inventors of U.S. Pat. Serial No. 09/434,845 and I conceived of the idea of user datagram

protocol (UDP) or response time reporter (RTR) jitter probes for measuring network performance including such criteria as data jitter, packet loss and round-trip latency. This idea included the subject matter of pending claims 4 and 21 of the application.

4. Attached as Exhibit A is a copy of an October 29, 1998 e-mail I sent to Frank Bordonaro, a co-inventor, which e-mail describes the algorithm for jitter, packet loss and round-trip latency (delay) measurement.

5. Continuously from at least October 29, 1998 until no later than December 30, 1998 by which time we had implemented, tested, logged and resolved a defect (attached as Exhibit B is a printout documenting this detected-and-fixed software bug), my co-inventors and I worked diligently to actually reduce to practice the invention described in the attached exhibits which includes the subject matter of pending claims 4 and 21.

6. Attached as Exhibit C is a copy of a formal software system design specification including the jitter packet (including a send time of day (TOD)) probe-based performance measure feature of the above e-mail dated January 4, 1999. It shows on page 2 that we checked the document into that system on January 11, 1999. This specification remains a part of Cisco Systems documentation control system.

I, the undersigned, declare that all statements made herein of my own knowledge are true, and that all statements made on information and belief are believed to be true; and further, that these statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United Stated Code, and that such willful false statements may jeopardize the validity of the application of any patent issuing thereon.

DATED this 26th day of June, 2003.

Kui Zhang

From kzhang Thu Oct 29 16:51:56 1998
Subject: design for jitter probe
To: fbordona (Frank Bordonaro)
Date: Thu, 29 Oct 1998 16:51:56 -0800 (PST)
Cc: sraparla (Rao Raparla), kzhang (Kui Zhang)
Reply-To: kzhang@cisco.com
X-Mailer: ELM [version 2.4 PL25]
MIME-Version: 1.0
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 7bit
Content-Length: 6398
Status: OR

This is the design for jitter probe Rao and I cooked up.

Design for RTR Jitter Probe

With the RTR Jitter probe, user can measure jitter, packet loss, as well as the round trip delay.

1. Jitter probe packet contains following fields (an expansion of current probe packet structure):

```
ushort probeType;
/* tells the responder what kind of probe this is */

ushort deltaTime;
/* time spent in the responder box */

sys_timestamp recv_timestamp;
/* receiver puts a timestamp when it receives the pacekt */

ushort send_seq_no;
/*
 * sequence number from the sender, this represents the number of packets
 * the sender has sent out so far
 */

ushort recv_seq_no;
/*
 * sequence number from the responder, this represents the number of
 * packets the responder has received so far
 */
```

We need at least 18 bytes in the UDP payload to carry the above fields. This is OK, since a VoIP packet is typically about 32 bytes in the UDP payload.

2. Operation:
   1). RTR sender tells the responder to start listen to the jitter probe packets via the RTR control protocol (just like current probes).

   2). Sender put the sequence number(starting from 1) in the send_seq_no field of the packet, then it sends it out. Sender keeps the timestamp when it sends out the packet. Sender does this for every interval. Everytime, the send_seq_no is incremented.

   3). RTR responder receives the packet, increments its receive counter.

It puts the receiving timestamp in the recv_timestamp field of the packet; puts the receive counter in the recv_seq_no field of the packet; calculates the time the packet spent in the responder box, put this delta time in the deltaTime field. Then it sends the packet back to the sender.

4). RTR sender receives the reply and saves the packet somewhere.

5). When RTR sender receives the next reply, it will compare the current reply to the last reply it receives. From these two packets, RTR will calculate the following:
(suppose these two packets are Pa and Pb)

```
Jitter = abs((recv_timestamp of Pb - recv_timestamp of Pa)
        - (send_timestamp of Pb - send_timestamp of Pa))
```

RTR will calculate jitter for all consecutive replies it receives, that is if Pb->send_seq_no - Pa->send_seq_no == 1. Jitter results will be saved into jitter distribution buckets, e.g., 4 buckets with size of 5ms will be like: bucket 1 for 0-5ms, 2 for 5-10ms, 3 for 10-15ms, 4 for 15ms and beyond.

RTR can calculate the packet loss from the last two packets it receives:

```
Loss(out) is packet Loss on the way to responder
Loss(in) is the packet loss on the way back from responder
Loss(out) = send_seq_no of Pb - recv_seq_no of Pb
Loss(in) = recv_seq_no of Pb - recv_seq_no of Pa - 1
```

If RTR does not get replies for the last severl packets it sends, e.g., RTR sends out 10 packets, but does not get reply for packet 9 and 10, then these two packets will be counted as MIA.

Total packet loss = Loss(out) + Loss(in) + MIA.

RTR can also calculate the round trip delay (just like existing udp echo probe, no news here).

3. Example

This example covers packet loss calculation. I found it is the most tricky to understand. Jitter calculation is quite straightforward.

1). RTR sender sends a packet with sequence number 1 to the responder.
2). Responder receives the packet, increment its recv counter to 1. then put the recv_seq_no as 1 in the reply packet.
3). Sender receives the packet, with send_seq_no == 1, recv_seq_no == 1. In a perfect world, send_seq_no == recv_seq_no
4). Sender sends packet 2, suppose this packet is lost on the way to the responder. Responder never gets it.
5). Sender sends packet 3, responder gets it, increment its recv counter to 2, so in the reply packet, send_seq_no == 3, recv_seq_no == 2.
6). Sender receives the reply with send_seq_no == 3, recv_seq_no == 2, Sender knows there is one packet lost on the way out (3-2). Sender also compares this packet with the previos reply it receives in step 3, send_seq_no==1, recv_seq_no==1. So there is no gap in the two recv_seq_no, this means there is no packet loss on the way back.

7). Sender send packet 4, responder got it, increment recv counter to 3.
    But the packet is lost on the way back.
8). Sender send packet 5, responder got it, increment recv counter to 4 then
    replies it.
9). Sender receives the reply: send_seq_no == 5, recv_seq_no == 4
    compare to the previous packet it receives in step 6,
    there is a gap between those two recv_seq_no, the previous recv_seq_no
    is 2. So there is one packet lost on the way back.
10). Sender send packet 6, no reply. Suppose this is the last packet sender
    wants to send. Sender has no way to know whether it is lost on the
    way out or on the way back. So this is labeled as MIA.
11). So the summary of packet loss:
    Loss(out) = 1
    Loss(in) = 1
    MIA = 1
    Total Loss = 3

4. CLI

```
rtr 1
type jitter destination-ip 9.0.0.1 destination-port 20 source-ip 1.0.0.1 \
   source-port 20 interval 10 packets 6 bucket 4 size 5
```

The above says:
Jitter probe to udp port 20 on host 9.0.0.1, with source ip address 1.0.0.1, source udp port 20, send 6 packets, with interval 10 ms. Jitter results will be saved in 4 buckets, with size of 5ms.

5. MIB

~~TBD. This needs to be carefully designed, since once a MIB is implemented,~~ it is hard to change. We need to consider other new probes when we design MIB for jitter.
Qwest incidcates they are willing to just use the CLI (they said they can write Perl scripts to extract numbers out of CLI output). So we don't have to complete this MIB within 4 weeks.

6. Testing

Testing is a little tricky for jitter measurement -- we need to somehow control the jitter of a network to test our measurement.
I'm thinking about modifying IOS to delay forwarding of certain packets, thus creating a controlled jitter.
i.e.
RTR sender -------Jitter inducing router ------RTR responder

CSCdk76846

Internally found moderate defect: Resolved (R)

rtr2:show rtr config doesnt reflect interval/num-packets

Full text of defect (~23K)
Summary of defect
View a new defect

Attachments: Description Diffs--rtr_phase2

Diffs--rtr_phase2: Added 981230 by kzhang

```
begin log-entry
 Subject: /vob/ios (branch rtr_phase2) Source Repository Modification
 kzhang      1998/12/30 23:04:23 UTC CSCdk76846 1998/12/30 15:04:23
local

 Comment   :
        Bugid used:
        CSCdk76846 rtr2:show rtr config doesnt reflect
interval/num-packets
        This commit also fixes:
        CSCdk76853: rtr2:PacketMIA does not reflect true # of MIA
packets
        CSCdk77291: rtr2:Min/Max of Positive/Negative SD/DS fields are
always 0
        CSCdk77935: rtr2:no indication when no route to jitter
responder

Elements :

/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_dump.c@@/main/connecticut/
v120t_3_pi/rtr_phase2/6

/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_jitter_probe.c@@/main/rtr_
phase2/5

/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_resp.c@@/main/v120t_3_pi/r
tr_phase2/2


end log-entry
#cmd=ccdiff -c
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_dump.c@@/main/connecticut/
v120t_3_pi/rtr_phase2/5
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_dump.c@@/main/connecticut/
```


EXHIBIT B

```
v120t_3_pi/rtr_phase2/6
Index: sys/rtt/rtt_dump.c
===================================================================
***
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_dump.c@@/main/connecticut/
v120t_3_pi/rtr_phase2/5       Tue Dec 22 17:48:43 1998
---
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_dump.c@@/main/connecticut/
v120t_3_pi/rtr_phase2/6       Wed Dec 30 15:05:15 1998
***************
*** 862,867 ****
--- 862,874 ----
           inRttMonEchoAdminEntry->rttMonEchoAdminPktDataRequestSize);
      printf("Response Size (ARR data portion): %u\n",

inRttMonEchoAdminEntry->rttMonEchoAdminPktDataResponseSize);
+     if (inRttMonEchoAdminEntry->rttMonEchoAdminNumPackets>0) {
+         printf("Num of Packets per probe: %u\n",
inRttMonEchoAdminEntry->rttMonEchoAdminNumPackets);
+     }
+     if (inRttMonEchoAdminEntry->rttMonEchoAdminInterval>0) {
+         printf("Interval between packets(milliseconds): %u\n",
+              inRttMonEchoAdminEntry->rttMonEchoAdminInterval);
+     }
      printf("Control Packets: ");
      if (inRttMonEchoAdminEntry->rttMonEchoAdminControlEnable ==
        D_rttMonEchoAdminControlEnable_true) {
***************
*** 993,1001 ****
--- 1000,1018 ----

  void outputJitterStats (rttJitterStats * jitterStats)
  {
+     printf("RTT Values:\n");
      printf("NumOfRTT: %u\t",jitterStats->NumOfRTT);
      printf("RTTSum: %u\t", jitterStats->RTTSum);
      printf("RTTSum2: %llu\n", jitterStats->RTTSum2);
+     printf("Packet Loss Values:\n");
+     printf("PacketLossSD: %u\t", jitterStats->PacketLossSD);
+     printf("PacketLossDS: %u\n", jitterStats->PacketLossDS);
+     printf("PacketOutOfSequence: %u\t",
jitterStats->PacketOutOfSequence);
+     printf("PacketMIA: %u\t", jitterStats->PacketMIA);
+     printf("PacketLateArrival: %u\n", jitterStats->PacketTimeOut);
+     printf("InternalError: %u\t", jitterStats->InternalError);
+     printf("Busies: %u\n", jitterStats->Busies);
+     printf("Jitter Values:\n");
      printf("MinOfPositivesSD: %u\t", jitterStats->MinOfPositivesSD);
```


EXHIBIT B

```
      printf("MaxOfPositivesSD: %u\n", jitterStats->MaxOfPositivesSD);
      printf("NumOfPositivesSD: %u\t", jitterStats->NumOfPositivesSD);
***************
*** 1016,1028 ****
      printf("NumOfNegativesDS: %u\t", jitterStats->NumOfNegativesDS);
      printf("SumOfNegativesDS: %u\t", jitterStats->SumOfNegativesDS);
      printf("Sum2NegativesDS: %llu\n", jitterStats->Sum2NegativesDS);
-     printf("PacketLossSD: %u\t", jitterStats->PacketLossSD);
-     printf("PacketLossDS: %u\n", jitterStats->PacketLossDS);
-     printf("PacketOutOfSequence: %u\t",
jitterStats->PacketOutOfSequence);
-     printf("PacketMIA: %u\t", jitterStats->PacketMIA);
-     printf("PacketTimeOut: %u\n", jitterStats->PacketTimeOut);
-     printf("InternalError: %u\t", jitterStats->InternalError);
-     printf("Busies: %u\n", jitterStats->Busies);
      return;
  }

--- 1033,1038 ----
#cmd=ccdiff -c
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_jitter_probe.c@@/main/rtr_
phase2/4
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_jitter_probe.c@@/main/rtr_
phase2/5
Index: sys/rtt/rtt_jitter_probe.c
===================================================================
***
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_jitter_probe.c@@/main/rtr_
phase2/4      Tue Dec 22 17:48:51 1998
---
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_jitter_probe.c@@/main/rtr_
phase2/5      Wed Dec 30 15:05:18 1998
***************
*** 38,51 ****
                              rttJitterStats * jitterStats)

  {
!
      inRttMonCtrlAdminQItem->curHourJitterStats->NumOfRTT
        += jitterStats->NumOfRTT;
      inRttMonCtrlAdminQItem->curHourJitterStats->RTTSum
        += jitterStats->RTTSum;
      inRttMonCtrlAdminQItem->curHourJitterStats->RTTSum2
        += jitterStats->RTTSum2;
!     if (jitterStats->MinOfPositivesSD
        <
inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesSD) {
        inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesSD
```


EXHIBIT B

```
            = jitterStats->MinOfPositivesSD;
--- 38,58 ----
                              rttJitterStats * jitterStats)

  {
!     if (jitterStats->InternalError == 1) {
!       inRttMonCtrlAdminQItem->curHourJitterStats->InternalError ++;
!       return (D_Return_Successfull);
!     }
!
      inRttMonCtrlAdminQItem->curHourJitterStats->NumOfRTT
        += jitterStats->NumOfRTT;
      inRttMonCtrlAdminQItem->curHourJitterStats->RTTSum
        += jitterStats->RTTSum;
      inRttMonCtrlAdminQItem->curHourJitterStats->RTTSum2
        += jitterStats->RTTSum2;
!     if
(inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesSD==0) {
!       inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesSD =
!           jitterStats->MinOfPositivesSD;
!     } else if (jitterStats->MinOfPositivesSD>0 &&
jitterStats->MinOfPositivesSD
        <
inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesSD) {
        inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesSD
            = jitterStats->MinOfPositivesSD;
***************
*** 62,69 ****
      inRttMonCtrlAdminQItem->curHourJitterStats->Sum2PositivesSD
        += jitterStats->Sum2PositivesSD;

!     if (jitterStats->MinOfNegativesSD
!       <
inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesSD) {
        inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesSD
            = jitterStats->MinOfNegativesSD;
      }
--- 69,79 ----
      inRttMonCtrlAdminQItem->curHourJitterStats->Sum2PositivesSD
        += jitterStats->Sum2PositivesSD;

!     if (inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesSD
== 0) {
!       inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesSD =
!           jitterStats->MinOfNegativesSD;
!     } else  if
(jitterStats->MinOfNegativesSD>0&&jitterStats->MinOfNegativesSD
!                 <
```

```
inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesSD) {
        inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesSD
            = jitterStats->MinOfNegativesSD;
      }
***************
*** 78,89 ****
        += jitterStats->SumOfNegativesSD;
      inRttMonCtrlAdminQItem->curHourJitterStats->Sum2NegativesSD
        += jitterStats->Sum2NegativesSD;
!
!     if (jitterStats->MinOfPositivesDS
        <
inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesDS) {
        inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesDS
            = jitterStats->MinOfPositivesDS;
      }
      if (jitterStats->MaxOfPositivesDS
        >inRttMonCtrlAdminQItem->curHourJitterStats->MaxOfPositivesDS)
{
        inRttMonCtrlAdminQItem->curHourJitterStats->MaxOfPositivesDS
--- 88,103 ----
        += jitterStats->SumOfNegativesSD;
      inRttMonCtrlAdminQItem->curHourJitterStats->Sum2NegativesSD
        += jitterStats->Sum2NegativesSD;
!
!     if (inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesDS
== 0) {
!       inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesDS =
!           jitterStats->MinOfPositivesDS;
!     } else if (jitterStats->MinOfPositivesDS>0 &&
jitterStats->MinOfPositivesDS
        <
inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesDS) {
        inRttMonCtrlAdminQItem->curHourJitterStats->MinOfPositivesDS
            = jitterStats->MinOfPositivesDS;
      }
+
      if (jitterStats->MaxOfPositivesDS
        >inRttMonCtrlAdminQItem->curHourJitterStats->MaxOfPositivesDS)
{
        inRttMonCtrlAdminQItem->curHourJitterStats->MaxOfPositivesDS
***************
*** 96,102 ****
      inRttMonCtrlAdminQItem->curHourJitterStats->Sum2PositivesDS
        += jitterStats->Sum2PositivesDS;

!     if (jitterStats->MinOfNegativesDS
        <
```


EXHIBIT B

```
inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesDS) {
	inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesDS
	    = jitterStats->MinOfNegativesDS;
--- 110,119 ----
      inRttMonCtrlAdminQItem->curHourJitterStats->Sum2PositivesDS
	+= jitterStats->Sum2PositivesDS;

!     if (inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesDS
== 0) {
!	inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesDS =
!	    jitterStats->MinOfNegativesDS;
!     } else if (jitterStats->MinOfNegativesDS>0 &&
jitterStats->MinOfNegativesDS
	<
inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesDS) {
	inRttMonCtrlAdminQItem->curHourJitterStats->MinOfNegativesDS
	    = jitterStats->MinOfNegativesDS;
***************
*** 149,158 ****
      ushort localPort;
      boolean hashAdded = FALSE;
      int length, lossIn, lossOut, jitterIn, jitterOut, MIA,
timeoutCounter;
!     int outOfOrderCounter;
      ushort sendSeqNo = 0;
      boolean done = FALSE;
      boolean loopMore;
      sys_timestamp endOfRecvTimer;

      lossIn = 0;
--- 166,176 ----
      ushort localPort;
      boolean hashAdded = FALSE;
      int length, lossIn, lossOut, jitterIn, jitterOut, MIA,
timeoutCounter;
!     int outOfOrderCounter, lastSendRecved;
      ushort sendSeqNo = 0;
      boolean done = FALSE;
      boolean loopMore;
+     boolean internalError = FALSE;
      sys_timestamp endOfRecvTimer;

      lossIn = 0;
***************
*** 167,172 ****
--- 185,191 ----
      sendTime = 0;
      recvTime = 0;
```

```
        preRecvTime = 0;
+       lastSendRecved = 0;
        memset(jitterStats, 0, sizeof(rttJitterStats));
        memset(ancillary, 0, RTT_ANCILLARY_SIZE);

***************
*** 205,214 ****

        localIP = sin.sin_addr.s_addr;

        sd = socket_open(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        if (sd == INVALID_FD) {
          RTT_BUGINF2(IDofMySelf, "\n\t socket create failed");
!         return;
        }

        /*
--- 224,235 ----

        localIP = sin.sin_addr.s_addr;

+       sd = -99;
        sd = socket_open(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        if (sd == INVALID_FD) {
          RTT_BUGINF2(IDofMySelf, "\n\t socket create failed");
!         internalError = TRUE;
!         goto end;
        }

        /*
***************
*** 217,222 ****
--- 238,244 ----

        if (socket_bind(sd, (sockaddr *) &sin, sizeof(sin)) == SO_FAIL)
{
            RTT_BUGINF2(IDofMySelf, "\n\t socket bind failed");
+         internalError = TRUE;
          goto end;
        }

***************
*** 224,229 ****
--- 246,252 ----
        length = sizeof(sin);
        if (socket_get_localname(sd, (sockaddr *) &sin, &length) ==
SO_FAIL) {
          RTT_BUGINF2(IDofMySelf, "\n\t socket bind failed: cannot get
```


EXHIBIT B

```
localname");
+       internalError = TRUE;
        goto end;
      }

***************
*** 240,245 ****
--- 263,269 ----
      status = socket_set_option(sd, SOL_IP, IP_PKTINFO, &on,
sizeof(on));
      if (status<0) {
          RTT_BUGINF2(IDofMySelf, "\n\t socket PKTINFO option setting
failed");
+       internalError = TRUE;
          goto end;
      }

***************
*** 289,294 ****
--- 313,319 ----

D_rttMonHistoryCollectionSense_notconnected,FALSE);
            RTT_BUGINF3(IDofMySelf,
                        "\n control message failure: %d", ctrlResult);
+           internalError = TRUE;
            goto end;
        }
      }
***************
*** 327,332 ****
--- 352,358 ----
                              probeArgs->DataRequestSize,
                              0, (sockaddr *) &sin, sizeof(sin)) < 0)
{
                RTT_BUGINF2(IDofMySelf, "\n\t socktest transmit: can't
send to server");
+               internalError = TRUE;
                goto end;
            }
            sendSeqNo++;
***************
*** 404,409 ****
--- 430,437 ----
                    continue;
                }

+               lastSendRecved = recvPak->sendSeqNo;
+
```


EXHIBIT B

```
                if (probeArgs->ControlEnable) {
                    /*
                     * if control protocol is enabled, we know the
other
***************
*** 427,433 ****
                                if
((unsigned)jitterOut>jitterStats->MaxOfPositivesSD) {
                                    jitterStats->MaxOfPositivesSD =
jitterOut;
                                }
!                               if
((unsigned)jitterOut<jitterStats->MinOfPositivesSD) {
                                    jitterStats->MinOfPositivesSD =
jitterOut;
                                }
                            } else if (jitterOut<0) {
--- 455,464 ----
                                if
((unsigned)jitterOut>jitterStats->MaxOfPositivesSD) {
                                    jitterStats->MaxOfPositivesSD =
jitterOut;
                                }
!                               if (jitterStats->MinOfPositivesSD ==
0) {
!                                   jitterStats->MinOfPositivesSD =
jitterOut;
!                               } else if ((unsigned)jitterOut <
!
jitterStats->MinOfPositivesSD && jitterOut>0) {
                                    jitterStats->MinOfPositivesSD =
jitterOut;
                                }
                            } else if (jitterOut<0) {
***************
*** 438,444 ****
                                if
((unsigned)jitterOut>jitterStats->MaxOfNegativesSD) {
                                    jitterStats->MaxOfNegativesSD =
jitterOut;
                                }
!                               if
((unsigned)jitterOut<jitterStats->MinOfNegativesSD) {
                                    jitterStats->MinOfNegativesSD =
jitterOut;
                                }
                            }
--- 469,477 ----
```


EXHIBIT B

```
                                  if
((unsigned)jitterOut>jitterStats->MaxOfNegativesSD) {
                                      jitterStats->MaxOfNegativesSD =
jitterOut;
                                  }
!                                 if (jitterStats->MinOfNegativesSD ==
0) {
!                                     jitterStats->MinOfNegativesSD =
jitterOut;
!                                 } else if
((unsigned)jitterOut<jitterStats->MinOfNegativesSD && jitterOut>0) {
                                      jitterStats->MinOfNegativesSD =
jitterOut;
                                  }
                              }
***************
*** 456,462 ****
                                  if
((unsigned)jitterIn>jitterStats->MaxOfPositivesDS) {
                                      jitterStats->MaxOfPositivesDS =
jitterIn;
                                  }
!                                 if
((unsigned)jitterIn<jitterStats->MinOfPositivesDS) {
                                      jitterStats->MinOfPositivesDS =
jitterIn;
                                  }
                              } else if (jitterIn<0) {
--- 489,497 ----
                                  if
((unsigned)jitterIn>jitterStats->MaxOfPositivesDS) {
                                      jitterStats->MaxOfPositivesDS =
jitterIn;
                                  }
!                                 if (jitterStats->MinOfPositivesDS ==
0) {
!                                     jitterStats->MinOfPositivesDS =
jitterIn;
!                                 } else if
((unsigned)jitterIn<jitterStats->MinOfPositivesDS && jitterIn>0) {
                                      jitterStats->MinOfPositivesDS =
jitterIn;
                                  }
                              } else if (jitterIn<0) {
***************
*** 467,473 ****
                                  if
((unsigned)jitterIn>jitterStats->MaxOfNegativesDS) {
```


EXHIBIT B

```
                                        jitterStats->MaxOfNegativesDS =
jitterIn;
                                    }
!                                   if
((unsigned)jitterIn<jitterStats->MinOfNegativesDS) {
                                        jitterStats->MinOfNegativesDS =
jitterIn;
                                    }

--- 502,510 ----
                                    if
((unsigned)jitterIn>jitterStats->MaxOfNegativesDS) {
                                        jitterStats->MaxOfNegativesDS =
jitterIn;
                                    }
!                                   if (jitterStats->MinOfNegativesDS==0)
{
!                                       jitterStats->MinOfNegativesDS =
jitterIn;
!                                   } else if
((unsigned)jitterIn<jitterStats->MinOfNegativesDS && jitterIn>0) {
                                        jitterStats->MinOfNegativesDS =
jitterIn;
                                    }

***************
*** 493,505 ****
        }
      } /* end of for loop */

      /* now we need to update stats here */
      memcpy(inRttMonCtrlAdminQItem->latestJitterStats, jitterStats,
           sizeof(rttJitterStats));
      updateJitterCaptureStats(inRttMonCtrlAdminQItem, jitterStats);

! end:
!     socket_close(sd);
      free(keepPak);
      free(jitterStats);
      free(ancillary);
--- 530,548 ----
        }
      } /* end of for loop */

+     MIA = probeArgs->numOfPkts - lastSendRecved;
+     jitterStats->PacketMIA = MIA;
+
+ end:
```


EXHIBIT B

```
+       if (internalError) {
+         jitterStats->InternalError = 1;
+       }
        /* now we need to update stats here */
        memcpy(inRttMonCtrlAdminQItem->latestJitterStats, jitterStats,
             sizeof(rttJitterStats));
        updateJitterCaptureStats(inRttMonCtrlAdminQItem, jitterStats);

!       if (sd != -99) socket_close(sd);
        free(keepPak);
        free(jitterStats);
        free(ancillary);
#cmd=ccdiff -c
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_resp.c@@/main/v120t_3_pi/r
tr_phase2/1
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_resp.c@@/main/v120t_3_pi/r
tr_phase2/2
Index: sys/rtt/rtt_resp.c
===================================================================
***
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_resp.c@@/main/v120t_3_pi/r
tr_phase2/1   Fri Dec 11 12:35:56 1998
---
/view/kzhang-rtr_phase2/vob/ios/sys/rtt/rtt_resp.c@@/main/v120t_3_pi/r
tr_phase2/2   Wed Dec 30 15:05:22 1998
***************
*** 974,982 ****
                    jitterPak->recvTime = pktinfo.icmp_inputtime;
                }
                delta = clock_icmp_time() - pktinfo.icmp_inputtime;
!               if (delta<0) delta = 0;
            } else {
                /* if for some reason, we cannot get it, use delta 0 */
                delta = 0;
            }

--- 974,986 ----
                    jitterPak->recvTime = pktinfo.icmp_inputtime;
                }
                delta = clock_icmp_time() - pktinfo.icmp_inputtime;
!               if (delta<0) {
!                   buginf("\ndelta cannot be less than 0");
!                   delta = 0;
!               }
            } else {
                /* if for some reason, we cannot get it, use delta 0 */
+               buginf("\nCannot get icmp timestamp on the packet");
                delta = 0;
```


EXHIBIT B

```
        }

***************
*** 1163,1169 ****
      rttRespEnable = TRUE;
      if (dispatch.pid == 0) {
        dispatch.pid = process_create(rtt_responder, "Rtt Responder",
!                                    NORMAL_STACK, PRIO_NORMAL);
      }
  }

--- 1167,1173 ----
      rttRespEnable = TRUE;
      if (dispatch.pid == 0) {
        dispatch.pid = process_create(rtt_responder, "Rtt Responder",
!                                    LARGE_STACK, PRIO_NORMAL);
      }
  }
```

CISCO SYSTEMS

Document Number	ENG-31226
Revision	B
Author	Rao Raparla
Project Manager	Frank Bordonaro

RTR II - Service Provider Enhancements

Software Unit Design Specification

Project Headline

Enhance IOS RTR features for Service Providers

EXHIBIT C

Reviewers

Department	Name
Development Enginnering	Frank Bordonaro
Development Test Engineering	Nga Vu
Customer Advocacy Representative	Kim Dion
Compliance Engineering	N/A
Knowledge Products	Mary Mangone, Amanda Worthington

Modification History

Rev.	Date	Originator	Comment
1.0	01/04/99	Rao Raparla	Draft
2.0	1/9/99	RTR Team	Update
2.1	2/23/99	John Lautmann	Update DHCP

Definitions

This section defines words, acronyms, and actions which may not be readily understood.

RTR	Response Time Reporter
HTTP	Hypertext Transfer Protocol
DNS	Domain Name Servvice
DLSw	Data Link Switching - see rfc 1795
DHCP	Dynamic Host Configuration Protocol
SSP	Switch to Switch packet for DLSw
DISCOVER	a broadcast frame looking for DHCP server
OFFER	a frame from a DHCP server with a proposed IP address for the client
LEASE	an IP address that lasts a fixed amount of time
BOUND	a client has accepted an IP address from the DHCP server

EXHIBIT C

1.0 Problem Definition

New service provider related Response Time Reporter functionality is being requested by customers such as Qwest, MCI, Sprint, AT&T, and NetCom. Additional Enterprise specific functionality is being requested by customers such as Glaxo, NationsBank, SONY, and IGN. To satisfy these requirements and retain a competitive advatage in the performance management market, the following new probe types willbe provided in the 12.0(5)T and 12.0(5)s release.

1.1 HTTP Probe

The purpose of the HTTP Probe is to measure the round trip time (RTT) taken to connect and access data from an HTTP server. A large amout of service provider and enterprise traffic is HTTP/TCP based, so this probe is critical in measuring network health for this specific type of user traffic.

__Measurements__

An HTTP query operation is preceded by location and connection establishment activities. RTR reports each of the measurements below as separate result metrics.

- **RTT taken to perform domain name service (DNS) lookup**

When an URL is specified, a DNS lookup must be made to resolve the hostname in URL to an IP address before the TCP connection to the HTTP server can be made. If the user doesn't specify a name-server, RTR will use the name-servers configured on the router. If name-server(s) are not configured, RTR will report an error. If an IP Address is specified in the URL(hence no need to resolve hostname), then DNS RTT is 0.

- **RTT taken to perform a TCP connect to the HTTP Server**

A TCP connection must be made to the HTTP server before any operation. The time take to make this connection is reported as a separate result.

- **RTT taken to perform HTTP operation**

The operations are define in the section below.

__Operation Modes__

The are two different types of modes for the HTTP probe.

- **GET-Mode**

In this mode, the user specifies the parameters required for an http GET operation and RTR will perform the GET operation, record RTT metrics outlined above as well as report the size of the page retrieved. This operation is very simple to use and is the most common type of HTTP performance measurement. In GET-mode, only the base html page will be retrieved, not the links contained in the page.

- **Raw-Mode**

Unlike GET-mode, where RTR is responsible for filling an HTTP request based on the URL specified by the user, the HTTP Raw mode requires the user to specify the entire content of an HTTP request. This will enable a user to send an HTTP request with any valid request header field, e.g, HEAD/POST requests or send a GET request with authentication information. For a POST operation, the user can specify up to 1K bytes of html text to put via the CLI or the MIB.

Similar to the GET-mode, RTR does not display the information that was received in the response expcept for the Status code and Message size.

__Web Caching__

EXHIBIT C

Many networks employ web caches to optimize network traffic. RTR allows for the use of web caches (by default) or allows a user to disable their use on a specific probe. RTR disables the use of a web cache by altering specific fields in the header field of the HTTP request. This can be specified by the following header fields:

1. Cache-Control = "no-cache" (in HTTP/1.1) (or)
2. Pragma = "no-cache" (in HTTP/1.0)

Note:Cache-Control directive is not necessarily implemented in HTTP/1.0 so we need to use pragma directive.

Uniform Resource Locator (URL)

For this release, RTR will only support a URL that references an "http" protocol. Support for the following URL format will be provided - http://<host>:<port>/<url-path>

Note that port number and url-path are optional. Examples for URL would be

"http://www.cisco.com/index.html"

"http://www.cisco.com/"

"http://www.cisco.com"

"http://www.mycom.com:80/myproduct.html"

Proxy HTTP Servers

If proxy http servers are used in a network, RTR will provide an option to specify the ip address of the proxy server when configuring the RTR probe. This will be the case for GET-Mode or Raw-Mode.

1.2 DNS Probe

The objective of this probe is to compute the round trip time (RTT) taken to query a DNS server. Response time is computed by taking the difference between the time taken to send the DNS request and receiving a reply. The probe queries for an IP address if the user specifies a hostname or queries for a hostname if the user specifies an IP Address.

1.3 UDP Jitter Probe

This probe is a superset of the UDP probe first releases in IOS version 12.0T(3). The 12.0T(3) version simply computes RTT of UDp packets sent with any 4-tuple source and destination of IP addresses and ports. The UDP jitter probe retains this capability and adds the measurement of per-direction inter-packet delay variance (jitter) and per-direction packet loss for the probe packets. Per-direction measurements indicate separately the packet-loss and jitter from the source-to-destination and destination-to-source.

Additionally, in this release, RTR will provide:

- Ability to set the IP precedence bits, known as type of service bits (ToS), for UDP or UDP-Jitter Probes.
- Support for asynchronous packet sizes

EXHIBIT C

1.4 DLSw Probe

DLSw is described in RFC1795. In short, it tunnels LAN protocols over a TCP/IP tunnel thereby extending the local LAN across a wide area network. It has become very popular and service providers sell this service to their enterprise customers. The two WAN attached routers that are translation points for the LAN to WAN mapping are called "peers". The LAN traffic that traverses the peers are group in entitites called "circuits". End to end circuit performance can be monitored by end stations, typically a PC, that are termination points for the LAN traffic. Currently there is no product that measures the resonse time between DLSw peers.

1.5 DHCP Probe

To allow for plug-and-play, many IP hosts learn their IP address at boot time. The DHCP (Dynamic Host Configuration Protocol - RFC-1541 and RFC-2131) is designed to provide this function. DHCP provides a mechanism for allocating IP address and other TCP/IP parameters dynamically. It also has a protocol for negotiating and transmitting host-specific information.

A DHCP server provides configuration information to requesting DHCP clients. A DHCP client uses DHCP protocol to discover its configuration and network address. A DHCP Proxy operates on behalf of a DHCP server(s) to provide configuration and network address to DHCP clients.

When a device wishes to obtain an IP address on a network, it first broadcasts a ***discover*** frame. Any available DHCP server sends back an ***offer*** frame. The device selects among one or more offers and ***binds*** an IP address. The ***lease*** lasts a fixed amount of time, such as one hour or one day. Currently there is no product that measure the amount of time needed to obtain an IP address.

2.0 Design Considerations

2.1 HTTP Probe

The HTTP Probe will use the existing RTR infrastructire for scheduling it's operation, however, since the probe reports multiple results, a new HTTP Results table must be created. As will all RTR probes with the exception of the UDP-based probes, the HTTP probes accuracy is affected by the CPU load on the source router and destination web server.

Traps and Thresholds

TBD.

2.2 DNS Probe

The DNS probe will use the existing RTR infrastructure for all its operations including saving the results. The DNS probe uses the existing IOS DNS code, however, a few functions were rewritten to fit the needs of the RTR.. These changes will be reviewed by the appropriate IOS group.

Traps and Thresholds

TBD.

EXHIBIT C

2.3 UDP Jitter Probe

As previosly mentioned, the UDP jitter prrobe allows the user can measure jitter, packet loss, as well as the round trip delay. This probes objective is to accurately measure the above metrics to allow customers to base service level agreements on the results. The amount of time a probe packet spends in the source and destination RTR routers is not counted in measurements which most accurately reflect the performance of the network.

Additional RTR Packet Fields

To accomplish this accuracy requirement and support the packet-loss and jitter functionality, the following fields are added to the RTR probe packet structure.

- Probe Type(2 bytes)
  Tells the responder what kind of probe this is. UDP echo probe is 1, UDP jitter probe is 2.
- Responder Processing Time(2 bytes)
  Time in miliseconds spent in the responder box
- Sender Timestamp(4 bytes)
  Sending router puts a timestmap when it sends the packet.
- Receiver Timestamp(4 bytes)
  Receiving router puts a timestamp when it receives the packet.
- Send Sequence Number(2 bytes)
  This field is set by the RTR sender and represents the number of packets the sender has sent out thus far during this instance of the probe.
- Receive Sequence Number(2 bytes)
  This field is set by the RTR responder and represents the number of packets the responder has received during this instance of the probe.

We need at least 16 bytes in the UDP payload to carry the above fields. This is OK, since a VoIP packet is typically about 32 bytes in the UDP payload.

Operation

1. RTR sender tells the responder to start listen to the jitter probe packets via the RTR control protocol (just like current probes).
2. Sender put the sequence number(starting from 1) in the send_seq_no field of the packet, it also puts a timestamp in the sending timestamp field, then it sends it out. Sender does this for every interval. Everytime, the send_seq_no is incremented.
3. RTR responder receives the packet, increments its receive counter. It puts the receiving timestamp in the recv_timestamp field of the packet; puts the receive counter in the recv_seq_no field of the packet; calculates the time the packet spent in the responder box, put this delta time in the deltaTime field. Then it sends the packet back to the sender.
4. RTR sender receives the reply and saves the packet.
5. When RTR sender receives the next reply, it will compare the current reply to the last reply it receives.
6. RTR jitter calculation. Suppose two consective packets are labeled as Pa and Pb. The jitter value from source to destination would be:

   Jitter = (Pb recv_time - Pa recv_time) - (Pb send_time - Pa send_time)

   RTR will calculate jitter for all consecutive replies it receives, that is if Pb->send_seq_no - Pa->send_seq_no == 1. Jitter results will be aggregated into sums, and sums of squares, so it is easy to calculate the average and standard deviations. Min and Max values will also be saved. Further more,

EXHIBIT C

positive jitter values and negative jitter values will be saved seperately, since they reflect different network conditions. Positive jitter value means increasing inter-arrival time on the destination, while negative jitter value means decreasing inter-arrival time.

Based on receive timestamp and responder processing time in the packets, we can also find out when the packet is send out from the responder. Thus we can calculate the jitter values from destination to the soruce, similarly as jitter from source to destination.

7. RTR packet-loss calculation
   a. Loss(SD) is packet Loss on the way to responder (from Source to Destination)
   b. **Loss(SD) = send_seq_no of Pb - recv_seq_no of Pb**
   c. Loss(DS) is the packet loss on the way back from responder (from Destination to Source)
   d. **Loss(DS) = recv_seq_no of Pb - recv_seq_no of Pa - 1**
   e. MIA: Missing In Action packets. If RTR does not get replies for the last several packets it sends, e.g., RTR sends out 10 packets, but does not get reply for packet 9 and 10, then these two packets will be counted as MIA, since we have no way to deduce if they are lost on the way out or on the way back.
   f. **Total packet loss = Loss(SD) + Loss(DS) + MIA**
8. RTR can also calculate the round trip delay (just like pre-existing udp echo probe).

Packet Loss Example

1. RTR sender sends a packet with sequence number 1 to the responder.
2. Responder receives the packet, increment its recv counter to 1. then put the recv_seq_no as 1 in the reply packet.
3. Sender receives the packet, with send_seq_no == 1, recv_seq_no == 1. In a perfect world, send_seq_no == recv_seq_no
4. Sender sends packet 2, suppose this packet is lost on the way to the responder. Responder never gets it.
5. Sender sends packet 3, responder gets it, increment its recv counter to 2, so in the reply packet, send_seq_no == 3, recv_seq_no == 2.
6. Sender receives the reply with send_seq_no == 3, recv_seq_no == 2, Sender knows there is one packet lost on the way out (3-2). Sender also compares this packet with the previos reply it receives in step 3, send_seq_no==1, recv_seq_no==1. So there is no gap in the two recv_seq_no, this means there is no packet loss on the way back.
7. Sender send packet 4, responder got it, increment recv counter to 3. But the packet is lost on the way back.
8. Sender send packet 5, responder got it, increment recv counter to 4 then replies it.
9. Sender receives the reply:send_seq_no == 5, recv_seq_no == 4 compare to the previous packet it receives in step 6, there is a gap between those two recv_seq_no, the previous recv_seq_no is 2. So there is one packet lost on the way back.
10. Sender send packet 6, no reply. Suppose this is the last packet sender wants to send. Sender has no way to know whether it is lost on the way out or on the way back. So this is labeled as MIA.
11. So the summary of packet loss:
    a. Loss(SD) = 1
    b. Loss(DS) = 1
    c. MIA = 1
    d. Total Loss = 3

New Data Structure To Keep Jitter Results

EXHIBIT C

Since jitter probe will report a number of values, we cannot use the existing RTR result data structure, which only keeps the round trip time value. A new MIB table will be created for jitter results.

Traps and Thresholds

TBD.

2.4 DLSw Probe

DLSw peers reside on a router uses the TCP/IP transport provided by IOS. The DLSw probe will send a packet over an existing DLSw connection. The packet will be a DLSw DLX packet of type "keepalive". This packet is normally sent from one DLSw peer to another to make ensure the other DLSw router is still healthy when no DLSw circuit level traffic is being sent for a period of time. In respnse to receiving a keepalive packet the other DLSw peer will immediately send a keepalive reply. The RTR process originating the keepalive packet will timestamp the packet in order to track the round-trip time it takes to send a DLSw packet. These probe keepalive packets are inserted into the normal DLSw stack processing and packet stream. The network can not distinguish DLSw keepalive packets from actual user or circuit traffic. Therefore, the DLSw probe accurately measure the perormance of the DLSw stack and underlying network.

RTR will leverage a registry function to request probe keepalive packets be inserted into the DLSw peer data stream. Dlsw code will hand RTR back the keepalive reply through a registry in the reverse direction. The DLSw code must keep a flag that indicates that the next keepalive reply should be routed to RTR. Note that if the DLSw peer already has a keepalive outstanding, the DLSw code should return a return code from the RTR registry call that indicates the service is unavailable at the present time.

For the RTR DLSw probe to return a response time, it is a prerequisite that the DLSw peers are in a connected state. For example, on router A, a probe can be defined for a DLSw peer B which doesn't need RTR in the image. The sending router should timestamp the packet upon delivery and upon receipt of the response. The registry call between RTR and DLSW will allow images with one but not both RTR and DLSw features to compile. The parser changes will be in the DLSw subsystem so that this probe can only be configured in images that also have DLSw.

Protocol Stacks

RTR - DLSw Probe

registry functions

DLSw

TCP driver

IP

Traps and Thresholds

Traps are sent for connection-loss, when the DLSw peer is not connected.

EXHIBIT C

2.5 DHCP Probe

RTR makes use of the DHCP client functionality of IOS. An IP address is obtained using the registry functions available in the DHCP client code. The address is immediately released.

There are two modes of DHCP probe operation: directed and non-directed. In the directed mode, a specific DHCP server is being tested. The IP address of the server must be entered into the router configuration.

In non-directed mode, the first available DHCP server is being tested. DHCP broadcasts ***discover*** frames on alll available interfaces. There may be multiple DHCP servers responding with ***offer*** frames. The first bound IP address is considered to be a completion. The time stored for that completion is time for the first bound IP address.

Traps and Thresholds

Traps are sent for timeout, when it takes longer than the pre-set timeout to obtain an IP lease. The threshold traps work similar to the existing echo threshold.

3.0 Functional Structure

Existing RTR software modules are being reused. For a description of the software modules see ENG-5398 and for a description of the RTR control and probe headers see ENG-24711.

~~4.0 System Flow~~

See ENG-5398 and ENG-24711.

5.0 Changes to RTR Link Protocol

The RTR control port protocol is unchanged by the addition of the functions outline in this spec.

The RTR UDP jitter packet contains the following fields:

```
struct rttJitterProbePak_ {
   ushort probeType;
   ushort deltaTime;
   ulong sendTime;
   ulong recvTime;
   ushort sendSeqNo;
   ushort recvSeqNo;
   uchar data[0];
};
```


EXHIBIT C

6.0 Algorithmic Description

See section 2.3 above for the UDP-Plus probe algorithm to measure jitter and packet-loss. No other non-obvious algorithms for the HTTP, DNS, DHCP, and DLSw probe implementations.

7.0 Interface Design

7.1 HTTP Probe

The HTTP probe uses the TCP sockets API. It does not need to use existing router HTTP APIs.

7.2 DNS Probe

The DNS probe uses IOS DNS API as needed.

7.3 UDP Jitter Probe

The UDP Probe uses the UDP sockets API. This API must be extended to provide RTR the ability to specifiy the IP ToS bits that are to be set on UDP datagrams.

7.4 DLSw Probe

A DLSw registry API has been defined to allow RTR to insert probe packets into the DLSw peer stream. Additionally, an RTR API has been defined to allow DLSw to hand reply packets back to RTR to complete the round trip path of the probe packets. The API's are as follows:

7.4.1 RTR Asking DLSw to Send a Probe

int return_code = reg_invoke_dlsw_rtr_request(destination peer ip address, payload_size in bytes);

Possible return codes: DLSw not running, peer not defined, peer not connected, can't be performed at the moment (due to a previous keepalive pending), probe sent.

7.4.2 DLSw Returning Probe Response

void reg_invoke_rtr_dlsw_response(peer ip address)

This API only gets called upon successful receipt on a response.

7.4.3 RTR Error Codes

One RTR MIB field will get updated with one of these values:

* DLSw not running
* Peer not defined
* Peer not connected
* Can't be performed at the moment (due to a previous keepalive pending)
* Timeout
* Success

7.5 DHCP Probe

The DHCP probe makes use of the DHCP Client functions. The functions include:

7.5.1 to obtain an IP address: reg_invoke_dhcpc_get_addr()

The registry returns one of the following values:

EXHIBIT C

DHCP_RET_FAIL - indicates a failure in getting an IP address

DHCP_RET_PENDING - indicates the request is still pending

DHCP_RET_OK - indicates an address was successfully bound

7.5.2 to release a bound IP address: reg_invoke_dhcpc_free_addr()

This is a void registry.

8.0 Memory and Performance Impact

TBD.

Estimate the additional memory requirements due to this feature.

Identify whether there is a performance impact, either positive or negative, due to this feature. Specify if there is a particular performance requirement or limitation for this feature.

Assess the complexity of the feature, and any necessary impact on existing features and user interface which might affect backwards compatibility.

9.0 Additional MIB design and implementation

MIB Extensions TBD.

Image packaging considerations TBD.

10.0 End User Interface

The following are a list of new commands that were created to support new probes.

10.1 HTTP Probe

In router configuration mode:

(config)# rtr *<number>*

(config-rtr)# type http operation *<type of operation>* **url** *<url>* **[name-server** *<ipaddress>*] **[version** *<version number>*] **[source-addr** *<name or ipaddr>*] **[source-port** *<port number>*] **[cache** *<enable | disable>*] **[proxy-url** *<url>*]

where type of operation is ***get*** for http get request or ***raw*** for user defined http request. For raw http request, there is another sub-mode within the entry mode

(config-rtr)# http-raw-request

(config-rtr-http-raw-requst)# *<text>*

(config-rtr-http-raw-requst)# *<text>*

(config-rtr-http-raw-requst)# exit

10.2 DNS probe

In router configuration mode:

(config)# rtr *<number>*

(config-rtr)# type dns target-addr *<target address>* **name-server** *<ipaddress>*

EXHIBIT C

10.3 CLI for UDP Jitter probe

In router configuration mode:

(config)# rtr *<number>*

(config-rtr)# type jitter dest-addr *<name or ipaddr>* **dest-port** *<port number>* **[source-addr** *<name or ipaddr>*] [**source-port** *<port number>*] [**control** *<enable | disable>*] **num-packets** *<number of packets>* **interval** *<inter-packet interval>*

10.4 DLSw probe

In router configuration mode:

(config)# rtr *<number>*

(config-rtr)# type dlsw peer-ip--addr *<ipaddress>*

(config-rtr)# [**request-data-size** *bytes*]

request-size range is 0-18k bytes.

10.5 DHCP Probe

In router configuration mode:

(config)# rtr *<number>*

(config-rtr)# type dhcp

To measure a specific DHCP server, identify one DHCP server in router configuration mode:

(config)# ip dhcp-server *<ipaddress>*

11.0 SW Restrictions and Configuration

A 500 probe maximum is enforced which includes any combination of RTR probe types..

12.0 FW Restrictions and Configuration

None.

13.0 HW Restrictions and Configuration

None.

14.0 External Restrictions and Configuration

RTR can run as a standalone router solution operating via teh command line interface. A more scalable method of operation is when RTR is controlled by the RTT MIB from a GUI based network performance application such as Cisco's Internetwork Performance Monitor (IPM) or Concord's Network Health.

15.0 Development Unit Testing

- UDP Jitter Probe Test Plan: ENG-30325
- HTTP Probe Test Plan: ENG-31065

EXHIBIT C

- DNS Probe Test Plan: ENG-31062
- DLSw Probe Test Plan: ENG-XXXXX
- DHCP Probe Test Plan: ENG-XXXXX

EXHIBIT C